

# Infinite horizon problems

tl;dr: Dynamic programming still works

**Cathy Wu**

6.7950 Reinforcement Learning: Foundations and Methods

# References

1. With many slides adapted from Alessandro Lazaric and Matteo Pirotta.
2. Dimitri P. Bertsekas. Dynamic Programming and Optimal Control. Volume 2. 4th Edition. (2012). Chapters 1-2: Discounted Problems.
3. R. E. Bellman. Dynamic Programming. Princeton University Press, Princeton, N.J., 1957.

# Outline

1. Infinite horizon Markov Decision Processes
2. Value iteration
3. Policy iteration

# Outline

- 1. Infinite horizon Markov Decision Processes**
  - a. Discounted, stochastic shortest path, average cost
  - b. Policies
  - c. Dynamic programming algorithm?
2. Value iteration
3. Policy iteration

# Example: The Amazing Goods Company

- *Description.* At each month  $t$ , a warehouse contains  $s_t$  items of a specific goods and the demand for that goods is  $D$  (stochastic). At the end of each month the manager of the warehouse can order  $a_t$  more items from the supplier.
- The **cost** of maintaining an inventory of  $s$  is  $h(s)$ .
- The **cost** to order  $a$  items is  $C(a)$ .
- The **income** for selling  $q$  items is  $f(q)$ .
- If the demand  $d \sim D$  is bigger than the available inventory  $s$ , customers that cannot be served leave.
- The **value of the remaining inventory** at the end of the year is  $g(s)$ .
- **Constraint:** the store has a maximum capacity  $C$ .



# Markov Decision Process

## Definition (Markov decision process)

A **Markov decision process** (MDP) is defined as a tuple  $M = (S, A, P, r, \gamma)$  where

- $S$  is the **state** space,  $\}$  often simplified to finite
- $A$  is the **action** space,  $\}$
- $P(s'|s, a)$  is the **transition probability** with  

$$P(s'|s, a) = \mathbb{P}(s_{t+1} = s' | s_t = s, a_t = a)$$
- $r(s, a, s')$  is the immediate **reward** at state  $s$  upon taking action  $a$ ,  $\}$  sometimes simply  $r(s)$
- $\gamma \in [0, 1)$  is the **discount factor**.

Example: The Amazing Goods Company

- **Discount:**  $\gamma = 0.95$ . A dollar today is worth more than a dollar tomorrow.

# Markov Decision Process

## Definition (Markov decision process)

A **Markov decision process** (MDP) is defined as a tuple  $M = (S, A, P, r, \gamma)$  where

- $S$  is the *state* space,
  - $A$  is the *action* space,
  - $P(s'|s, a)$  is the *transition probability* with
 
$$P(s'|s, a) = \mathbb{P}(s_{t+1} = s' | s_t = s, a_t = a)$$
  - $r(s, a, s')$  is the immediate *reward* at state  $s$  upon taking action  $a$ ,
  - $\gamma \in [0, 1)$  is the *discount factor*.
- } often simplified to finite  
} sometimes simply  $r(s)$

Example: The Amazing Goods Company

- **Objective:**  $V(s_0; a_0, \dots) = \sum_{t=0}^{\infty} \gamma^t r_t$ . This corresponds to the cumulative reward, including the value of the remaining inventory at “the end.”
- The “**horizon**” of the problem is 12 (12 months in 1 year), i.e.  $r_{12} = g(s_{12})$ ;  $r_t = 0, t > 12$ .

# Markov Decision Process

## Definition (Markov decision process)

A **Markov decision process** (MDP) is defined as a tuple  $M = (S, A, P, r, \gamma)$  where

- $S$  is the *state* space,
- $A$  is the *action* space,
- $P(s'|s, a)$  is the *transition probability* with

$$P(s'|s, a) = \mathbb{P}(s_{t+1} = s' | s_t = s, a_t = a)$$

- $r(s, a, s')$  is the immediate *reward* at state  $s$  upon taking action  $a$ ,
- $\gamma \in [0, 1)$  is the *discount factor*.

☞ Two missing ingredients:

- How actions are selected → *policy*.
- What determines which actions (and states) are good → *value function*.



# Policy

## Definition (Policy)

A **decision rule**  $d$  can be

- **Deterministic**:  $d: S \rightarrow A$ ,
- **Stochastic**:  $d: S \rightarrow \Delta(A)$ ,
- **History-dependent**:  $d: H_t \rightarrow A$ ,
- **Markov**:  $d: S \rightarrow \Delta(A)$ ,

A **policy** (strategy, plan) can be

- **Stationary**:  $\pi = (d, d, d, \dots)$ ,
- (More generally) **Non-stationary**:  $\pi = (d_0, d_1, d_2, \dots)$

👉 For simplicity, we will typically write  $\pi$  instead of  $d$  for stationary policies, and  $\pi_t$  instead of  $d_t$  for non-stationary policies.

# Recall: The Amazing Goods

## Company Example

- *Description.* At each month  $t$ , a warehouse contains  $s_t$  *items* of a specific goods and the demand for that goods is  $D$  (stochastic). At the end of each month the manager of the warehouse can *order*  $a_t$  more items from the supplier.
- The **cost** of maintaining an inventory of  $s$  is  $h(s)$ .
- The **cost** to order  $a$  items is  $C(a)$ .
- The **income** for selling  $q$  items is  $f(q)$ .
- If the demand  $d \sim D$  is bigger than the available inventory  $s$ , customers that cannot be served leave.
- The **value of the remaining inventory** at the end of the year is  $g(s)$ .
- **Constraint:** the store has a maximum capacity  $C$ .



# Recall: The Amazing Goods

## Company Example

- Description. At each month  $t$ , a warehouse contains  $s_t$  items of a specific goods and the demand for that goods is  $D$  (stochastic). At the end of each month the manager of the warehouse can order  $a_t$  more items from the supplier.
- The cost of maintaining an inventory of  $s$  is  $h(s)$ .
- The cost to order  $a$  items is  $C(a)$ .
- The income for selling  $q$  items is  $f(q)$ .
- If the demand  $d \sim D$  is bigger than the available inventory  $s$ , customers that cannot be served leave.
- The value of the remaining inventory at the end of the year is  $g(s)$ .
- Constraint:** the store has a maximum capacity  $C$ .



Stationary policy composed of deterministic Markov decision rules

$$\pi(s) = \begin{cases} C - s & \text{if } s < M/4 \\ 0 & \text{otherwise} \end{cases}$$

# Recall: The Amazing Goods

## Company Example

- Description. At each month  $t$ , a warehouse contains  $s_t$  items of a specific goods and the demand for that goods is  $D$  (stochastic). At the end of each month the manager of the warehouse can order  $a_t$  more items from the supplier.
- The cost of maintaining an inventory of  $s$  is  $h(s)$ .
- The cost to order  $a$  items is  $C(a)$ .
- The income for selling  $q$  items is  $f(q)$ .
- If the demand  $d \sim D$  is bigger than the available inventory  $s$ , customers that cannot be served leave.
- The value of the remaining inventory at the end of the year is  $g(s)$ .
- Constraint:** the store has a maximum capacity  $C$ .



Stationary policy composed of stochastic history-dependent decision rules

$$\pi(s_t) = \begin{cases} U(C - s_{t-1}, C - s_{t-1} + 10) & \text{if } s_t < s_{t-1}/2 \\ 0 & \text{otherwise} \end{cases}$$

# Optimization Problem

- Our goal: solve the MDP

## Definition (Optimal policy and optimal value function)

The solution to an MDP is an **optimal policy**  $\pi^*$  satisfying

$$\pi^* \in \arg \max_{\pi \in \Pi} V^\pi$$

where  $\Pi$  is some policy set of interest.

The corresponding value function is the **optimal value function**

$$V^* = V^{\pi^*}$$

# State Value Function

- Given a policy  $\pi = (d_1, d_2, \dots)$  (deterministic to simplify notation)
  - **Infinite time horizon with discount**: the problem never terminates but rewards which are **closer** in time receive a **higher** importance.

$$V^\pi(s) = \mathbb{E} \left[ \sum_{t=0}^{\infty} \gamma^t r(s_t, \pi_t(h_t)) \mid s_0 = s; \pi \right]$$

with discount factor  $0 \leq \gamma < 1$ :

- **Small** = short-term rewards, **big** = long-term rewards
  - For any  $\gamma \in [0, 1)$  the series always converges (for bounded rewards)
- **Used when**: there is uncertainty about the deadline and/or an intrinsic definition of discount.

# State Value Function

- Given a policy  $\pi = (d_1, d_2, \dots)$  (deterministic to simplify notation)
  - **Finite time horizon  $T$** : deadline at time  $T$ , the agent focuses on the sum of the rewards up to  $T$ .

$$V^\pi(t, s) = \mathbb{E} \left[ \sum_{\tau=t}^{T-1} r(s_\tau, \pi_\tau, (h_\tau)) + R(s_T) \mid s_t = s; \pi = (\pi_t, \dots, \pi_T) \right]$$

where  $R$  is a value function for the final state.

- **Used when**: there is an intrinsic deadline to meet.

# State Value Function

- Given a policy  $\pi = (d_1, d_2, \dots)$  (deterministic to simplify notation)
  - **Stochastic shortest path**: the problem never terminates but the agent will eventually reach a **termination state**.

$$V^\pi(s) = \mathbb{E} \left[ \sum_{t=0}^T r(s_t, \pi_t, (h_t)) \mid s_0 = s; \pi \right]$$

where  $T$  is the first (**random**) time when the **termination state** is achieved.

- **Used when**: there is a specific goal condition.



# State Value Function

- Given a policy  $\pi = (d_1, d_2, \dots)$  (deterministic to simplify notation)
  - **Infinite time horizon with average reward**: the problem never terminates but the agent only focuses on the (expected) **average of the rewards**.

$$V^\pi(s) = \lim_{T \rightarrow \infty} \mathbb{E} \left[ \frac{1}{T} \sum_{t=0}^{T-1} r(s_t, \pi_t, (h_t)) \mid s_0 = s; \pi \right]$$

- **Used when**: the system should be constantly controlled over time.

## *Notice*

From now on we mostly work in the *discounted infinite horizon* setting (except Lecture 5).

Most results (*not always so smoothly*) extend to other settings.

# DP applies to infinite horizon problems, too!

- Finite horizon **stochastic** and **Markov** problems (e.g. driving, robotics, games)

$$\begin{aligned} V_T^*(s_T) &= r_T(s_T) \quad \text{for all } s_T \quad (\text{terminal reward}) \\ V_t^*(s_t) &= \max_{a_t \in A} r_t(s_t, a_t) + \mathbb{E}_{s_{t+1} \sim P(\cdot | s_t, a_t)} [V_{t+1}^*(s_{t+1})] \\ &\text{for all } s_t, \text{ and } t = \{T - 1, \dots, 0\} \end{aligned}$$

- **From finite to (discounted) infinite horizon problems?**
- **Infinite** horizon stochastic problems (e.g. package delivery over months or years, long-term customer satisfaction, control of autonomous vehicles)

$$V^*(s) = \max_{a \in A} r(s, a) + \gamma \mathbb{E}_{s' \sim P(\cdot | s, a)} [V^*(s')] \quad \text{for all } s$$

# Really?

- **Infinite** horizon stochastic problems (e.g. package delivery over months or years, long-term customer satisfaction, control of autonomous vehicles)

$$V^*(s) = \max_{a \in A} r(s, a) + \gamma \mathbb{E}_{s' \sim P(\cdot | s, a)} [V^*(s')] \quad \text{for all } s$$

👉 This is called the **optimal Bellman equation**.

- An **optimal policy** is such that:

$$\pi^*(s) = \underset{a \in A}{\text{arg max}} \left[ r(s, a) + \gamma \mathbb{E}_{s' \sim P(\cdot | s, a)} [V^*(s')] \right] \quad \text{for all } s$$

- **Discuss:** Any difficulties with this new algorithm?

# Outline

1. Infinite horizon Markov Decision Processes
2. **Value iteration**
  - a. Bellman operators, Optimal Bellman equation, and properties
  - b. Convergence
  - c. Numerical example
3. Policy iteration

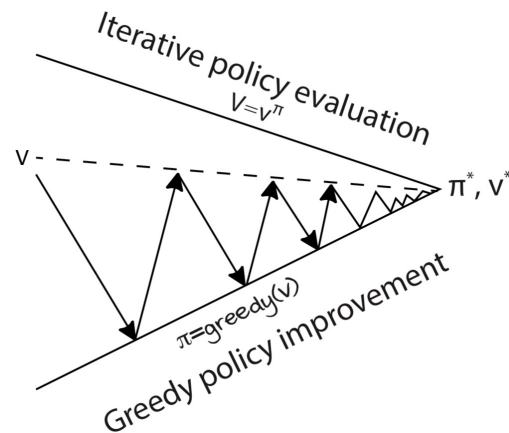
# Value iteration algorithm

1. Let  $V_0(s)$  be **any function**  $V_0: S \rightarrow \mathbb{R}$ . [Note: not stage 0, but iteration 0.]
2. Apply the **principle of optimality** so that given  $V_i$  at iteration  $i$ , we compute
$$V_{i+1}(s) = \mathcal{T}V_i(s) = \max_{a \in A} r(s, a) + \gamma \mathbb{E}_{s' \sim P(\cdot|s,a)} [V_i(s')] \quad \text{for all } s$$
3. Terminate when  $V_i$  stops improving, e.g. when  $\max_s |V_{i+1}(s) - V_i(s)|$  is small.
4. Return the greedy policy:  $\pi_K(s) = \arg \max_{a \in A} r(s, a) + \gamma \mathbb{E}_{s' \sim P(\cdot|s,a)} V_K(s')$

☞ A key result:  $V_i \rightarrow V^*$ , as  $i \rightarrow \infty$ .

## ☞ Helpful properties

- Markov process
- Contraction in max-norm
- Cauchy sequences
- Fixed point



Adapted from Morales, Grokking Deep Reinforcement Learning, 2020.

# Value iteration algorithm

1. Let  $V_0(s)$  be **any function**  $V_0: S \rightarrow \mathbb{R}$ . [Note: not stage 0, but iteration 0.]
2. Apply the **principle of optimality** so that given  $V_i$  at iteration  $i$ , we compute
$$V_{i+1}(s) = \mathcal{T}V_i(s) = \max_{a \in A} r(s, a) + \gamma \mathbb{E}_{s' \sim P(\cdot|s, a)} [V_i(s')] \quad \text{for all } s$$
3. Terminate when  $V_i$  stops improving, e.g. when  $\max_s |V_{i+1}(s) - V_i(s)|$  is small.
4. Return the greedy policy:  $\pi_K(s) = \arg \max_{a \in A} r(s, a) + \gamma \mathbb{E}_{s' \sim P(\cdot|s, a)} V_K(s')$

## Definition (Optimal Bellman operator)

For any  $W \in \mathbb{R}^{|S|}$ , the optimal Bellman operator is defined as

$$\mathcal{T}W(s) = \max_{a \in A} r(s, a) + \gamma \mathbb{E}_{s' \sim P(\cdot|s, a)} W(s') \quad \text{for all } s$$

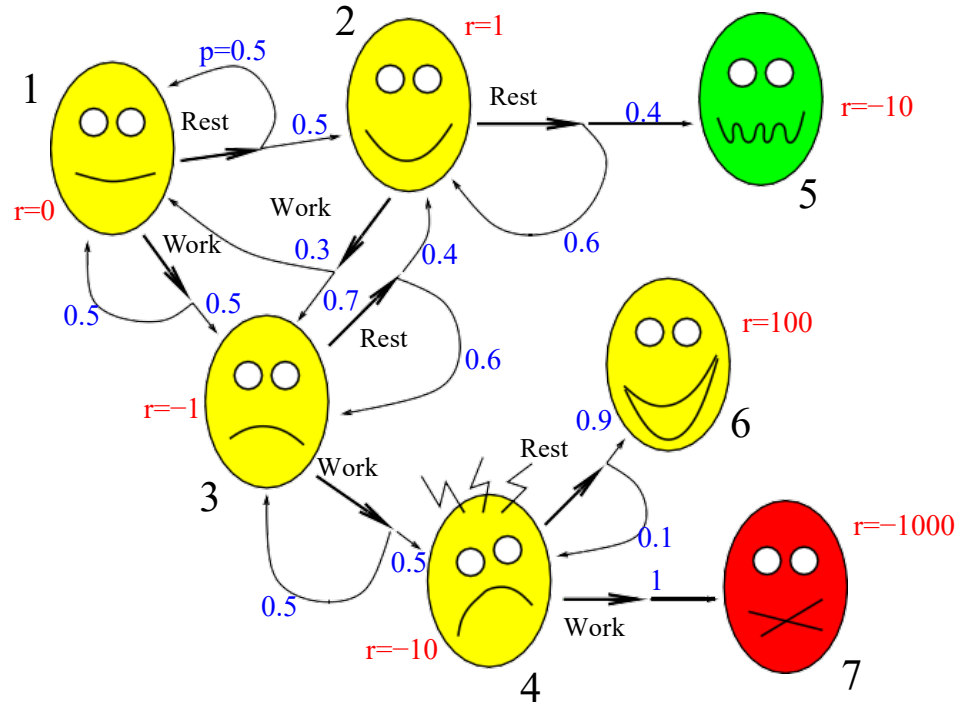
☞ Then we can write the algorithm step **2** concisely:

$$V_{i+1}(s) = \mathcal{T}V_i(s) \quad \text{for all } s$$

**Key question:** Does  $V_i \rightarrow V^*$ ?

# The student dilemma

- *Model*: all the transitions are Markov, states  $s_5, s_6, s_7$  are terminal.
- *Setting*: infinite horizon with terminal states.
- *Objective*: find the policy that maximizes the expected sum of rewards before achieving a terminal state.
- *Notice*: Not a discounted infinite horizon setting. But the Bellman equations hold unchanged.
- **Discuss**: What kind of problem setting is this? (Hint: value function.)





## *The Optimal Bellman Equation*

### **Bellman's Principle of Optimality** (Bellman (1957)):

*“An **optimal policy** has the property that, whatever the initial state and the initial decision are, the remaining decisions must constitute an **optimal policy** with regard to the **state resulting from the first decision.**”*

# The Optimal Bellman Equation

## Theorem (Optimal Bellman Equation)

The optimal value function  $V^*$  (i.e.  $V^* = \max_{\pi} V^{\pi}$ ) is the solution to the optimal Bellman equation:

$$V^*(s) = \max_{a \in A} \left[ r(s, a) + \gamma \sum_{s'} p(s'|s, a) V^*(s') \right]$$

And any optimal policy is such that:

$$\pi^*(a|s) \geq 0 \Leftrightarrow a \in \arg \max_{a' \in A} \left[ r(s, a') + \gamma \sum_{s'} p(s'|s, a') V^*(s') \right]$$

Or, for short:  $V^* = \mathcal{T}V^*$

☞ There is always a deterministic policy (see: Puterman, 2005, Chapter 7)

# Proof: The Optimal Bellman Equation

For any policy  $\pi = (a, \pi')$  (possibly non-stationary),

$$\begin{aligned} V^*(s) &= \max_{\pi} \mathbb{E} \left[ \sum_{t \geq 0} \gamma^t r(s_t, \pi(s_t)) \mid s_0 = s; \pi \right] && \text{[value function]} \\ &= \max_{(a, \pi')} \left[ r(s, a) + \gamma \sum_{s'} p(s'|s, a) V^{\pi'}(s') \right] && \text{[Markov property \& change of "time"]} \\ &= \max_a \left[ r(s, a) + \gamma \sum_{s'} p(s'|s, a) \max_{\pi'} V^{\pi'}(s') \right] \\ &= \max_a \left[ r(s, a) + \gamma \sum_{s'} p(s'|s, a) V^*(s') \right] && \text{[value function]} \end{aligned}$$

# Proof: Line 2 (also, the Bellman Equation)

For simplicity, consider any stationary policy  $\pi = (\pi, \pi, \dots)$ ,

$$\begin{aligned} V^\pi(s) &= \mathbb{E} \left[ \sum_{t=0}^{\infty} \gamma^t r(s_t, \pi(s_t)) \mid s_0 = s; \pi \right] && \text{[value function]} \\ &= r(s, \pi(s)) + \mathbb{E} \left[ \sum_{t=1}^{\infty} \gamma^t r(s_t, \pi(s_t)) \mid s_0 = s; \pi \right] && \text{[Markov property]} \\ &= r(s, \pi(s)) + \gamma \sum_{s'} \mathbb{P}(s_1 = s' \mid s_0 = s; \pi(s_0)) \mathbb{E} \left[ \sum_{t=1}^{\infty} \gamma^{t-1} r(s_t, \pi(s_t)) \mid s_1 = s'; \pi \right] \\ &&& \text{[MDP and change of "time"]} \\ &= r(s, \pi(s)) + \gamma \sum_{s'} p(s' \mid s, \pi(s)) \mathbb{E} \left[ \sum_{t'=0}^{\infty} \gamma^{t'} r(s_{t'}, \pi(s_{t'})) \mid s_{0'} = s'; \pi \right] \\ &= r(s, \pi(s)) + \gamma \sum_{s'} p(s' \mid s, \pi(s)) V^\pi(s') && \text{[value function]} \end{aligned}$$

## Proof: Line 3

For the =, we have:

$$\max_{\pi'} \sum_{s'} p(s'|s, a) V^{\pi'}(s') \leq \sum_{s'} p(s'|s, a) \max_{\pi'} V^{\pi'}(s')$$

But, let  $\bar{\pi}(s') = \operatorname{argmax}_{\pi'} V^{\pi'}(s')$

$$\sum_{s'} p(s'|s, a) \max_{\pi'} V^{\pi'}(s') \leq \sum_{s'} p(s'|s, a) V^{\bar{\pi}}(s') \leq \max_{\pi'} \sum_{s'} p(s'|s, a) V^{\pi'}(s')$$

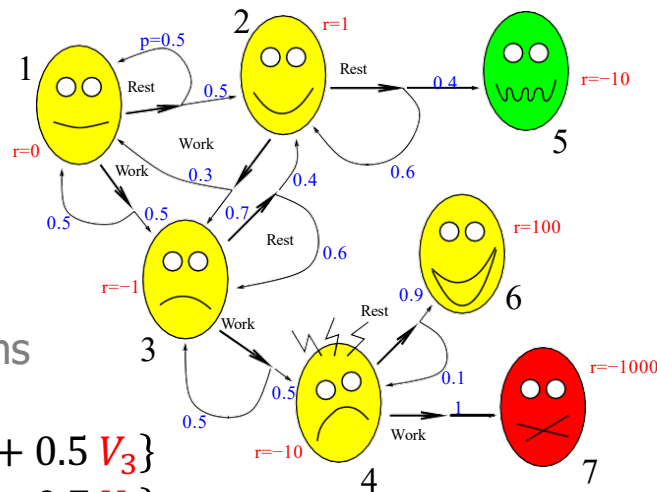


# The student dilemma

$$V^*(s) = \max_{a \in A} \left[ r(x, a) + \gamma \sum_y p(y|x, a) V^*(y) \right]$$

System of equations

$$\begin{cases} V_1 = \max\{0 + 0.5 V_1 + 0.5 V_2; 0 + 0.5 V_1 + 0.5 V_3\} \\ V_2 = \max\{0 + 0.4 V_5 + 0.6 V_2; 0 + 0.3 V_1 + 0.7 V_3\} \\ V_3 = \max\{-1 + 0.4 V_2 + 0.6 V_3; -1 + 0.5 V_4 + 0.5 V_3\} \\ V_4 = \max\{-10 + 0.9 V_6 + 0.1 V_4; -10 + V_7\} \\ V_5 = -10 \\ V_6 = 100 \\ V_7 = -1000 \end{cases}$$



**Discuss:** How to solve this system of equations?

# System of Equations

The optimal Bellman equation:

$$V^*(s) = \max_{a \in A} \left[ r(s, a) + \gamma \sum_{s'} p(s'|s, a) V^*(s') \right]$$

Is a **non-linear** system of equations with  $N$  unknowns and  $N$  non-linear constraints (i.e. the **max** operator).

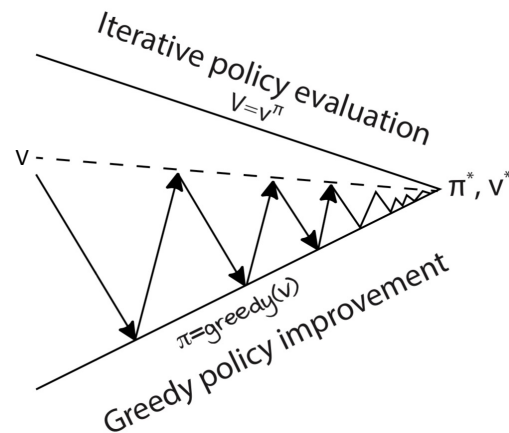
# Value iteration algorithm

1. Let  $V_0(s)$  be **any function**  $V_0: S \rightarrow \mathbb{R}$ . [Note: not stage 0, but iteration 0.]
2. Apply the **principle of optimality** so that given  $V_i$  at iteration  $i$ , we compute
$$V_{i+1}(s) = \mathcal{T}V_i(s) = \max_{a \in A} r(s, a) + \gamma \mathbb{E}_{s' \sim P(\cdot|s,a)} [V_i(s')] \quad \text{for all } s$$
3. Terminate when  $V_i$  stops improving, e.g. when  $\max_s |V_{i+1}(s) - V_i(s)|$  is small.
4. Return the greedy policy:  $\pi_K(s) = \arg \max_{a \in A} r(s, a) + \gamma \mathbb{E}_{s' \sim P(\cdot|s,a)} V_K(s')$

☞ A key result:  $V_i \rightarrow V^*$ , as  $i \rightarrow \infty$ .

## ☞ Helpful properties

- Markov process
- Contraction in max-norm
- Cauchy sequences
- Fixed point



Adapted from Morales, Grokking Deep Reinforcement Learning, 2020.



# Properties of Bellman Operators

## Proposition

1. **Contraction in  $L_\infty$ -norm**: for any  $W_1, W_2, \in \mathbb{R}^N$

$$\|\mathcal{T}W_1 - \mathcal{T}W_2\|_\infty \leq \gamma \|W_1 - W_2\|_\infty$$

2. **Fixed point**:  $V^*$  is the **unique fixed point** of  $\mathcal{T}$ , i.e.  $V^* = \mathcal{T}V^*$ .

Proof: value iteration

- From **contraction** property of  $\mathcal{T}$ ,  $V_k = \mathcal{T}V_{k-1}$ , and optimal value function  $V^* = \mathcal{T}V^*$ :

$$\begin{aligned} & \|V^* - V_{k+1}\|_\infty \\ &= \|\mathcal{T}V^* - \mathcal{T}V_k\|_\infty && \text{[optimal Bellman eq. and value iteration]} \\ &\leq \gamma \|V^* - V_k\|_\infty && \text{[contraction]} \\ &\leq \gamma^{k+1} \|V^* - V_0\|_\infty && \text{[recursion]} \\ &\rightarrow 0 \\ V_k &\rightarrow V^* && \text{[fixed point]} \end{aligned}$$

# Properties of Bellman Operators

## Proposition

1. **Contraction in  $L_\infty$ -norm:** for any  $W_1, W_2, \in \mathbb{R}^N$

$$\|\mathcal{T}W_1 - \mathcal{T}W_2\|_\infty \leq \gamma \|W_1 - W_2\|_\infty$$

2. **Fixed point:**  $V^*$  is the **unique fixed point** of  $\mathcal{T}$ , i.e.  $V^* = \mathcal{T}V^*$ .

Proof: value iteration

- **Convergence rate.** Let  $\epsilon > 0$  and  $\|r\|_\infty \leq r_{\max}$ , then after at most

$$\|V^* - V_k\|_\infty \leq \gamma^k \|V^* - V_0\|_\infty < \epsilon \implies K > \frac{\log\left(\frac{r_{\max}}{(1-\gamma)\epsilon}\right)}{\log\left(\frac{1}{\gamma}\right)}$$

# Proof: Contraction of the Bellman Operator

For any  $s \in S$

$$|\mathcal{T}W_1(s) - \mathcal{T}W_2(s)|$$

$$= \left| \max_a \left[ r(s, a) + \gamma \sum_{s'} p(s'|s, a) W_1(s') \right] - \max_{a'} \left[ r(s, a') + \gamma \sum_{s'} p(s'|s, a') W_2(s') \right] \right|$$

$$\leq \max_a \left| \left[ r(s, a) + \gamma \sum_{s'} p(s'|s, a) W_1(s') \right] - \left[ r(s, a) + \gamma \sum_{s'} p(s'|s, a) W_2(s') \right] \right|$$

$$= \gamma \max_a \sum_{s'} p(s'|s, a) |W_1(s') - W_2(s')|$$

$$\leq \gamma \|W_1 - W_2\|_\infty \max_a \sum_{s'} p(s'|s, a) = \gamma \|W_1 - W_2\|_\infty$$



$$\max_x f(x) - \max_{x'} g(x') \leq \max_x (f(x) - g(x))$$

# Value Iteration: the Guarantees

## Corollary

Let  $V_K$  be the function computed after  $K$  iterations by value iteration, then the greedy policy

$$\pi_K(s) \in \arg \max_{a \in A} \left[ r(s, a) + \gamma \sum_{s'} p(s'|s, a) V_K(s') \right]$$

is such that

$$\underbrace{\|V^* - V^{\pi_K}\|_\infty}_{\text{performance loss}} \leq \frac{2\gamma}{1-\gamma} \underbrace{\|V^* - V_K\|_\infty}_{\text{approx. error}}$$

Furthermore, there exists  $\epsilon > 0$  such that if  $\|V^* - V_K\|_\infty \leq \epsilon$ , then  $\pi_K$  is **optimal**.

# Proof: Performance Loss

- **Note 1:** We drop the  $K$  everywhere.
- **Note 2:**  $\pi$  is the greedy policy corresponding to  $V$ , and  $V^\pi$  is the value function evaluated with  $\pi$ .

$$\begin{aligned}\|V^* - V^\pi\|_\infty &\leq \|TV^* - T^\pi V\|_\infty + \|T^\pi V - T^\pi V^\pi\|_\infty \\ &\leq \|TV^* - TV\|_\infty + \gamma\|V - V^\pi\|_\infty \\ &\leq \gamma\|V^* - V\|_\infty + \gamma(\|V - V^*\|_\infty + \|V^* - V^\pi\|_\infty) \\ &\leq \frac{2\gamma}{1-\gamma}\|V^* - V\|_\infty\end{aligned}$$



# Value Iteration: the Complexity

## Time complexity

- Each iteration takes on the order of  $S^2A$  operations.

$$V_{k+1}(s) = \mathcal{T}V_k(s) = \max_{a \in A} \left[ r(s, a) + \gamma \sum_{s'} p(s'|s, a) V_k(s') \right]$$

- The computation of the greedy policy takes on the order of  $S^2A$  operations.

$$\pi_K(s) \in \arg \max_{a \in A} \left[ r(s, a) + \gamma \sum_{s'} p(s'|s, a) V_K(s') \right]$$

- Total time complexity on the order of  $KS^2A$ .

## Space complexity

- Storing the MDP: dynamics on the order of  $S^2A$  and reward on the order of  $SA$ .
- Storing the value function and the optimal policy on the order of  $S$ .

# Value Iteration: Extensions and Implementations

## Asynchronous VI:

1. Let  $V_0$  be any vector in  $R^N$
2. At each iteration  $k = 1, 2, \dots, K$ 
  - Choose a state  $s_k$
  - Compute  $V_{k+1}(s_k) = TV_k(s_k)$

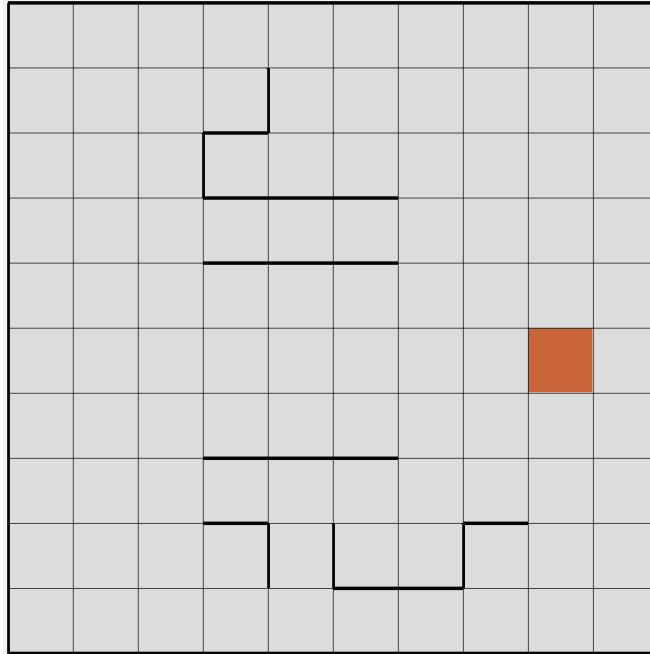
3. Return the greedy policy

$$\pi_K(s) \in \arg \max_{a \in A} \left[ r(s, a) + \gamma \sum_{s'} p(s'|s, a) V_K(s') \right]$$

## Comparison

- Reduced time complexity to  $O(SA)$
- Using round-robin, number of iterations increased by at most  $O(KS)$  but much smaller in practice if states are properly prioritized
- Convergence guarantees if no **starvation**

# The Grid-World Problem

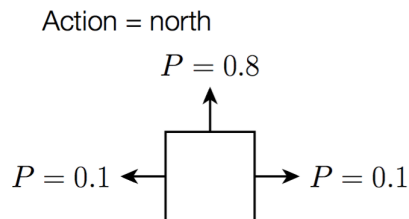




# Example: Winter parking (with ice and potholes)

- Simple grid world with a *goal state* (green, desired parking spot) with reward (+1), a *“bad state”* (red, pothole) with reward (-100), and all other states neural (+0).
- *Omnidirectional vehicle (agent)* can head in any direction. Actions move in the desired direction with probably 0.8, in one of the perpendicular directions with.
- Taking an action that would bump into a wall leaves agent where it is.

0	0	0	1
0		0	-100
0	0	0	0



[Source: adapted from Kolter, 2016]

# Example: value iteration

Running value iteration with  $\gamma = 0.9$

0	0	0	1
0		0	-100
0	0	0	0

Original reward function

(a)

Recall value iteration algorithm:

$$V_{i+1}(s) = \max_{a \in A} r(s, a) + \gamma \mathbb{E}_{s' \sim P(\cdot | s, a)} V_i(s') \quad \text{for all } s$$

Let's arbitrarily initialize  $V_0$  as the reward function, since it can be any function.

Example update (red state):

$$V_1(\text{red}) = -100 + \gamma \max \left\{ \begin{array}{ll} 0.8V_0(\text{green}) + 0.1V_0(\text{red}) + 0, & [\text{up}] \\ 0 + 0.1V_0(\text{red}) + 0, & [\text{down}] \\ 0 + 0.1V_0(\text{green}) + 0, & [\text{left}] \\ 0.8V_0(\text{red}) + 0.1V_0(\text{green}) + 1 & \} [\text{right}] \end{array} \right.$$

$$= -100 + 0.9(0.1 * 1) = -99.91 \text{ [best: go left]}$$

# Example: value iteration

Running value iteration with  $\gamma = 0.9$

0	0	0	1
0		0	-100
0	0	0	0

Original reward function

(a)

Recall value iteration algorithm:

$$V_{i+1}(s) = \max_{a \in A} r(s, a) + \gamma \mathbb{E}_{s' \sim P(\cdot | s, a)} V_i(s') \quad \text{for all } s$$

Let's arbitrarily initialize  $V_0$  as the reward function, since it can be any function.

Example update (green state):

$$V_1(\text{red}) = 1 + \gamma \max \left\{ \begin{array}{l} 0.8V_0(\text{green}) + 0.1V_0(\text{green}), \quad [\text{up}] \\ 0.8V_0(\text{red}) + 0.1V_0(\text{green}), \quad [\text{down}] \\ 0 + 0.1V_0(\text{green}) + 0.1V_0(\text{red}), \quad [\text{left}] \\ 0.8V_0(\text{red}) + 0.1V_0(\text{green}) + 0 \quad \} \quad [\text{right}]$$

$$= 1 + 0.9(0.9 * 1) = 1.81 \quad [\text{best: go up}]$$

# Example: value iteration

Running value iteration with  $\gamma = 0.9$

0	0	0	1
0	■	0	-100
0	0	0	0

Original reward function

(a)

Running value iteration with  $\gamma = 0.9$

0	0	0.72	1.81
0	■	0	-99.91
0	0	0	0

$\hat{V}$  at one iteration

(b)

Recall value iteration algorithm:

$$V_{i+1}(s) = \max_{a \in A} r(s, a) + \gamma \mathbb{E}_{s' \sim P(\cdot | s, a)} V_i(s') \quad \text{for all } s$$

Let's arbitrarily initialize  $V_0$  as the reward function, since it can be any function.

Need to also do this for all the "unnamed" states, too.

# Example: value iteration

Running value iteration with  $\gamma = 0.9$

0	0	0	1
0		0	-100
0	0	0	0

Original reward function

(a)

Running value iteration with  $\gamma = 0.9$

0	0	0.72	1.81
0		0	-99.91
0	0	0	0

$\hat{V}$  at one iteration

(b)

Running value iteration with  $\gamma = 0.9$

0.809	1.598	2.475	3.745
0.268		0.302	-99.59
0	0.034	0.122	0.004

$\hat{V}$  at five iterations

(c)

Running value iteration with  $\gamma = 0.9$

2.686	3.527	4.402	5.812
2.021		1.095	-98.82
1.390	0.903	0.738	0.123

$\hat{V}$  at 10 iterations

(d)

Running value iteration with  $\gamma = 0.9$

5.470	6.313	7.190	8.669
4.802		3.347	-96.67
4.161	3.654	3.222	1.526

$\hat{V}$  at 1000 iterations

(e)

Running value iteration with  $\gamma = 0.9$

→	→	→	↑
↑		←	←
↑	←	←	↓

Resulting policy after 1000 iterations

(f)

# Outline

1. Infinite horizon Markov Decision Processes
2. Value iteration
3. **Policy iteration**
  - a. Bellman equation, and properties
  - b. Convergence
  - c. Geometric interpretations
  - d. Generalized policy iteration

# More generally...

## Value iteration:

1.  $V_{i+1}(s) = \max_{a \in A} r(s, a) + \gamma \mathbb{E}_{s' \sim P(\cdot|s,a)} [V_i(s')] \quad \text{for all } s$
2.  $\pi_K(s) = \arg \max_{a \in A} r(s, a) + \gamma \mathbb{E}_{s' \sim P(\cdot|s,a)} V_K(s')$

## Related Operations:

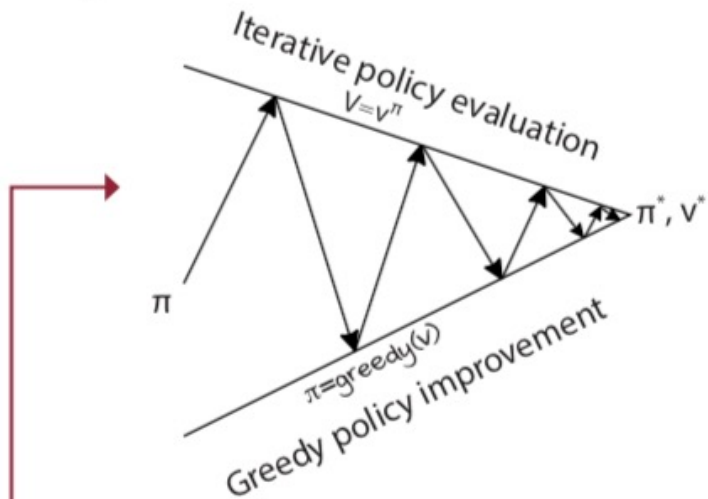
- Policy evaluation:  $V_{i+1}(s) = r(s, \pi_i(s)) + \gamma \mathbb{E}_{s' \sim P(\cdot|s, \pi_i(s))} [V_i(s')] \quad \text{for all } s$
- Policy improvement:  $\pi_i(s) = \arg \max_{a \in A} r(s, a) + \gamma \mathbb{E}_{s' \sim P(\cdot|s,a)} V_i(s')$

## 👉 Generalized Policy Iteration:

- Repeat:
  1. Policy evaluation for  $N$  steps
  2. Policy improvement
- Value iteration:  $N = 1$ ; Policy iteration:  $N = \infty$

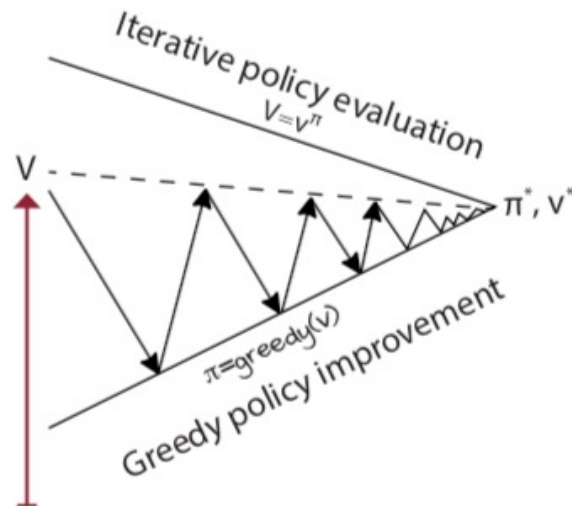
# In pictures

## Policy iteration



(1) Policy iteration consists of a full convergence of iterative policy evaluation alternating with greedy policy improvement.

## Value iteration



(2) value iteration starts with an arbitrary value function and has a truncated policy evaluation step.



# Policy Iteration: the Idea

1. Let  $\pi_0$  be any stationary policy

2. At each iteration  $k = 1, 2, \dots, K$

- Policy evaluation: given  $\pi_k$ , compute  $V^{\pi_k}$
- Policy improvement: compute the greedy policy

$$\pi_{k+1}(s) \in \arg \max_{a \in A} \left[ r(s, a) + \gamma \sum_{s'} p(s'|s, a) V^{\pi_k}(s') \right]$$

3. Stop if  $V^{\pi_k} = V^{\pi_{k-1}}$

4. Return the last policy  $\pi_K$

# Policy Iteration: the Guarantees

## Proposition

The policy iteration algorithm generates a sequence of policies with non-decreasing performance

$$V^{\pi_{k+1}} \geq V^{\pi_k}$$

and it converges to  $\pi^*$  in a finite number of iterations.

# The Bellman Equation

## Theorem (Bellman equation)

For any stationary policy  $\pi = (\pi, \pi, \dots)$ , at any state  $s \in S$ , the state value function satisfies the Bellman equation:

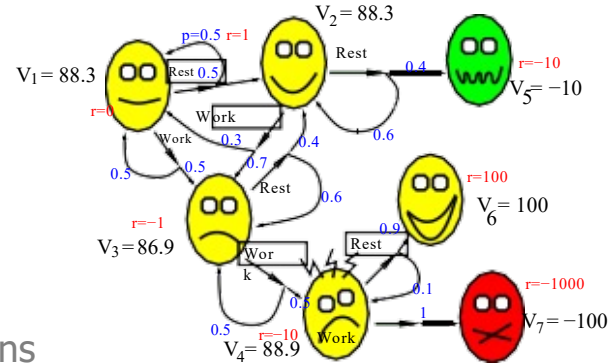
$$V^\pi(s) = r(s, \pi(s)) + \gamma \sum_{s' \in S} p(s'|s, \pi(s)) V^\pi(s')$$

# The student dilemma

- Discuss: How to solve this system of equations?

$$V^\pi(x) = r(x, \pi(x)) + \gamma \sum_y p(y|x, \pi(x)) V^\pi(y)$$

System of equations



$$\begin{cases} V_1 = 0 + 0.5 V_1 + 0.5 V_2 \\ V_2 = 1 + 0.3 V_1 + 0.7 V_3 \\ V_3 = -1 + 0.5 V_4 + 0.5 V_3 \\ V_4 = -10 + 0.9 V_6 + 0.1 V_4 \\ V_5 = -10 \\ V_6 = 100 \\ V_7 = -1000 \end{cases} \Rightarrow$$

$$V, R \in \mathbb{R}^7, P^\pi \in \mathbb{R}^{7 \times 7}$$

$$V = R + PV$$

⇓

$$V = (I - P)^{-1}R$$

# Recap: The Bellman Operators

Notation. w.l.o.g. a discrete state space  $|S| = N$  and  $V^\pi \in \mathbb{R}^N$   
(analysis extends to include  $N \rightarrow \infty$ )

## Definition

For any  $W \in \mathbb{R}^N$ , the Bellman operator  $T^\pi: \mathbb{R}^N \rightarrow \mathbb{R}^N$  is

$$T^\pi W(s) = r(s, \pi(s)) + \gamma \sum_{s'} p(s'|s, \pi(s)) W(s')$$

And the optimal Bellman operator (or dynamic programming operator) is

$$TW(s) = \max_{a \in A} \left[ r(s, a) + \gamma \sum_{s'} p(s'|s, a) W(s) \right]$$

# The Bellman Operators

## Proposition

Properties of the Bellman operators

1. Monotonicity: For any  $W_1, W_2 \in \mathbb{R}^N$ , if  $W_1 \leq W_2$  component-wise, then

$$T^\pi W_1 \leq T^\pi W_2$$

$$TW_1 \leq TW_2$$

2. Offset: For any scalar  $c \in \mathbb{R}$ ,

$$T^\pi(W - cI_N) = T^\pi W + \gamma c I_N$$

$$T(W - cI_N) = TW + \gamma c I_N$$

# The Bellman Operators

## Proposition

3. Contraction in  $L_\infty$ -norm: For any  $W_1, W_2 \in \mathbb{R}^N$

$$\|T^\pi W_1 - T^\pi W_2\|_\infty \leq \gamma \|W_1 - W_2\|_\infty$$

$$\|TW_1 - TW_2\|_\infty \leq \gamma \|W_1 - W_2\|_\infty$$

4. Fixed point: For any policy  $\pi$ ,

$V^\pi$  is the **unique fixed point** of  $T^\pi$

$V^*$  is the **unique fixed point** of  $T$

- For any  $W \in \mathbb{R}^N$  and any stationary policy  $\pi$

$$\lim_{k \rightarrow \infty} (T^\pi)^k W = V^\pi$$

$$\lim_{k \rightarrow \infty} (T)^k W = V^*$$

# Policy Iteration: the Idea

1. Let  $\pi_0$  be any stationary policy

2. At each iteration  $k = 1, 2, \dots, K$

- Policy evaluation: given  $\pi_k$ , compute  $V^{\pi_k}$
- Policy improvement: compute the greedy policy

$$\pi_{k+1}(s) \in \arg \max_{a \in A} \left[ r(s, a) + \gamma \sum_{s'} p(s'|s, a) V^{\pi_k}(s') \right]$$

3. Stop if  $V^{\pi_k} = V^{\pi_{k-1}}$

4. Return the last policy  $\pi_K$



# Policy Iteration: the Guarantees

## Proposition

The policy iteration algorithm generates a sequence of policies with non-decreasing performance

$$V^{\pi_{k+1}} \geq V^{\pi_k}$$

and it converges to  $\pi^*$  in a finite number of iterations.

# Proof: Policy Iteration

- From the definition of the Bellman operators and the greedy policy  $\pi_{k+1}$

$$V^{\pi_k} = \mathcal{J}^{\pi_k} V^{\pi_k} \leq \mathcal{J} V^{\pi_k} = \mathcal{J}^{\pi_{k+1}} V^{\pi_k}$$

- and from the monotonicity property of  $\mathcal{J}^{\pi_{k+1}}$ , it follows that

$$\begin{aligned} V^{\pi_k} &\leq \mathcal{J}^{\pi_{k+1}} V^{\pi_k} \\ \mathcal{J}^{\pi_{k+1}} V^{\pi_k} &\leq (\mathcal{J}^{\pi_{k+1}})^2 V^{\pi_k} \end{aligned}$$

$$\begin{aligned} &\dots \\ (\mathcal{J}^{\pi_{k+1}})^{n-1} V^{\pi_k} &\leq (\mathcal{J}^{\pi_{k+1}})^n V^{\pi_k} \\ &\dots \end{aligned}$$

- Joining all inequalities in the chain, we obtain

$$V^{\pi_k} \leq \lim_{n \rightarrow \infty} (\mathcal{J}^{\pi_{k+1}})^n V^{\pi_k} = V^{\pi_{k+1}}$$

- Then  $(V^{\pi_k})_k$  is a non-decreasing sequence.

# Policy Iteration: the Guarantees

Since a finite MDP admits a finite number of policies, then the termination condition is eventually met for a specific  $k$ .

Thus eq. 1 holds with an equality and we obtain

$$V^{\pi_k} = \mathcal{J}V^{\pi_k}$$

and  $V^{\pi_k} = V^*$  which implies that  $\pi_k$  is an optimal policy.



# Policy Iteration: Complexity

Notation. For any policy  $\pi$  the reward vector is  $r^\pi(x) = r(x, \pi(x))$  and the transition matrix is  $[P^\pi]_{x,y} = p(y|x, \pi(x))$

- Policy Evaluation Step

- Direct computation: For any policy  $\pi$  compute

$$V^\pi = (I - \gamma P^\pi)^{-1} r^\pi$$

**Complexity:**  $O(S^3)$ .

- Iterative policy evaluation: For any policy  $\pi$

$$\lim_{n \rightarrow \infty} T^\pi V_0 = V^\pi$$

**Complexity:** An  $\epsilon$ -approximation of  $V^\pi$  requires  $O\left(S^2 \frac{\log(\frac{1}{\epsilon})}{\log(\frac{1}{\gamma})}\right)$  steps.

- Monte-Carlo simulation: In each state  $s$ , simulate  $n$  trajectories  $\left((s_t^i)_{t \geq 0}\right)_{1 \leq i \leq n}$  following policy  $\pi$  and compute

$$\hat{V}^\pi(s) \simeq \frac{1}{n} \sum_{i=1}^n \sum_{t \geq 0} \gamma^t r(s_t^i, \pi(s_t^i))$$

**Complexity:** In each state, the approximation error is  $O\left(\frac{r_{\max}}{1-\gamma} \sqrt{\frac{1}{n}}\right)$ .

# Policy Iteration: Complexity

- Policy Improvement Step
  - Complexity  $O(S^2A)$
  
- Number of Iterations
  - At most  $O\left(\frac{SA}{1-\gamma} \log\left(\frac{1}{1-\gamma}\right)\right)$
  - Other results exist that do not depend on  $\gamma$

# Comparison between Value and Policy Iteration

## ■ Value Iteration

- **Pros:** each iteration is **computationally efficient**.
- **Cons:** convergence is only **asymptotic**.

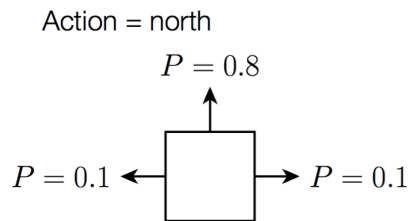
## ■ Policy Iteration

- **Pros:** converge in a **finite** number of iterations (often small in practice).
- **Cons:** each iteration requires a full **policy evaluation** and it might be expensive.

# Example: Winter parking (with ice and potholes)

- Simple grid world with a *goal state* (green, desired parking spot) with reward (+1), a *“bad state”* (red, pothole) with reward (-100), and all other states neural (+0).
- *Omnidirectional vehicle (agent)* can head in any direction. Actions move in the desired direction with probably 0.8, in one of the perpendicular directions with.
- Taking an action that would bump into a wall leaves agent where it is.

0	0	0	1
0		0	-100
0	0	0	0



[Source: adapted from Kolter, 2016]

# Example: value iteration

Running value iteration with  $\gamma = 0.9$

0	0	0	1
0		0	-100
0	0	0	0

Original reward function

(a)

Running value iteration with  $\gamma = 0.9$

0	0	0.72	1.81
0		0	-99.91
0	0	0	0

$\hat{V}$  at one iteration

(b)

Running value iteration with  $\gamma = 0.9$

0.809	1.598	2.475	3.745
0.268		0.302	-99.59
0	0.034	0.122	0.004

$\hat{V}$  at five iterations

(c)

Running value iteration with  $\gamma = 0.9$

2.686	3.527	4.402	5.812
2.021		1.095	-98.82
1.390	0.903	0.738	0.123

$\hat{V}$  at 10 iterations

(d)

Running value iteration with  $\gamma = 0.9$

5.470	6.313	7.190	8.669
4.802		3.347	-96.67
4.161	3.654	3.222	1.526

$\hat{V}$  at 1000 iterations

(e)

Running value iteration with  $\gamma = 0.9$

→	→	→	↑
↑		←	←
↑	←	←	↓

Resulting policy after 1000 iterations

(f)



# Example: policy iteration

Running policy iteration with  $\gamma = 0.9$ , initialized with policy  $\pi(s) = \text{North}$

0	0	0	1
0		0	-100
0	0	0	0

Original reward function

(a)

Running policy iteration with  $\gamma = 0.9$ , initialized with policy  $\pi(s) = \text{North}$

0.418	0.884	2.331	6.367
0.367		-8.610	-105.7
-0.168	-4.641	-14.27	-85.05

$V^\pi$  at one iteration

(b)

Running policy iteration with  $\gamma = 0.9$ , initialized with policy  $\pi(s) = \text{North}$

5.414	6.248	7.116	8.634
4.753		2.881	-102.7
2.251	1.977	1.849	-8.701

$V^\pi$  at two iterations

(c)

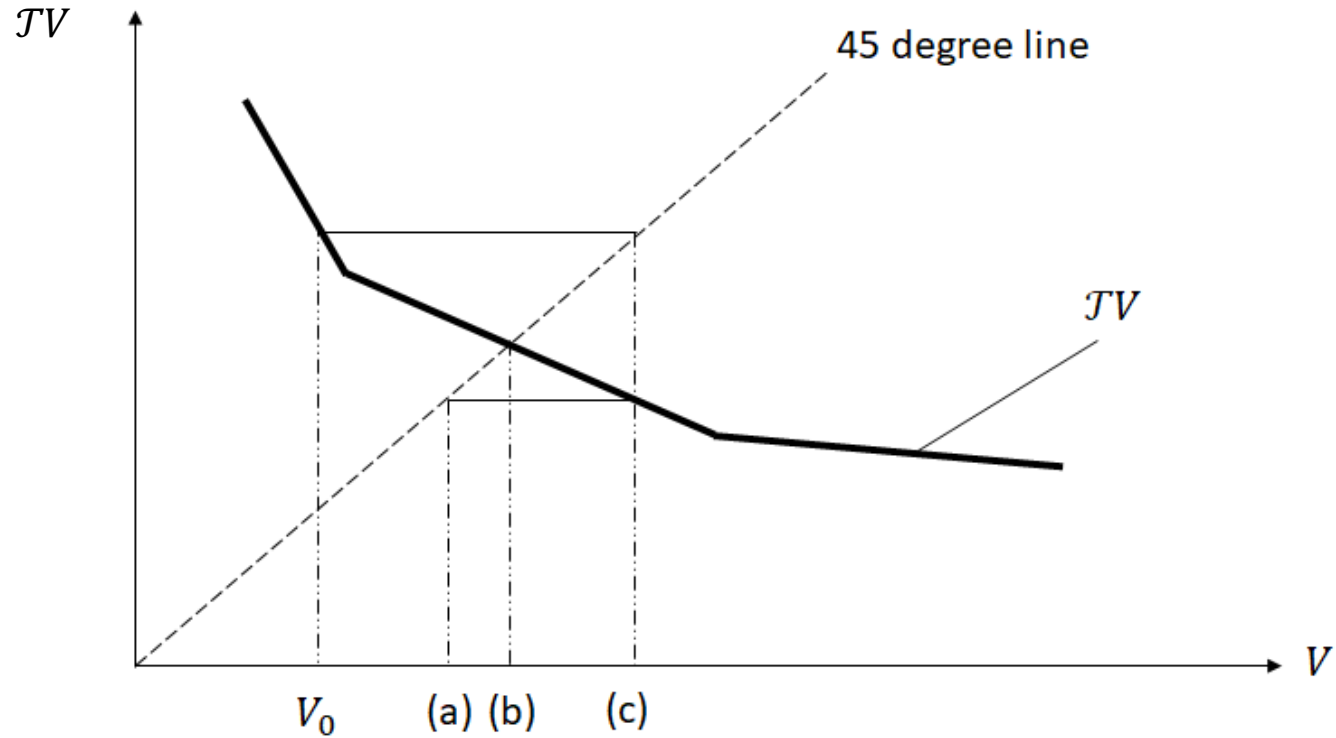
Running policy iteration with  $\gamma = 0.9$ , initialized with policy  $\pi(s) = \text{North}$

5.470	6.313	7.190	8.669
4.803		3.347	-96.67
4.161	3.654	3.222	1.526

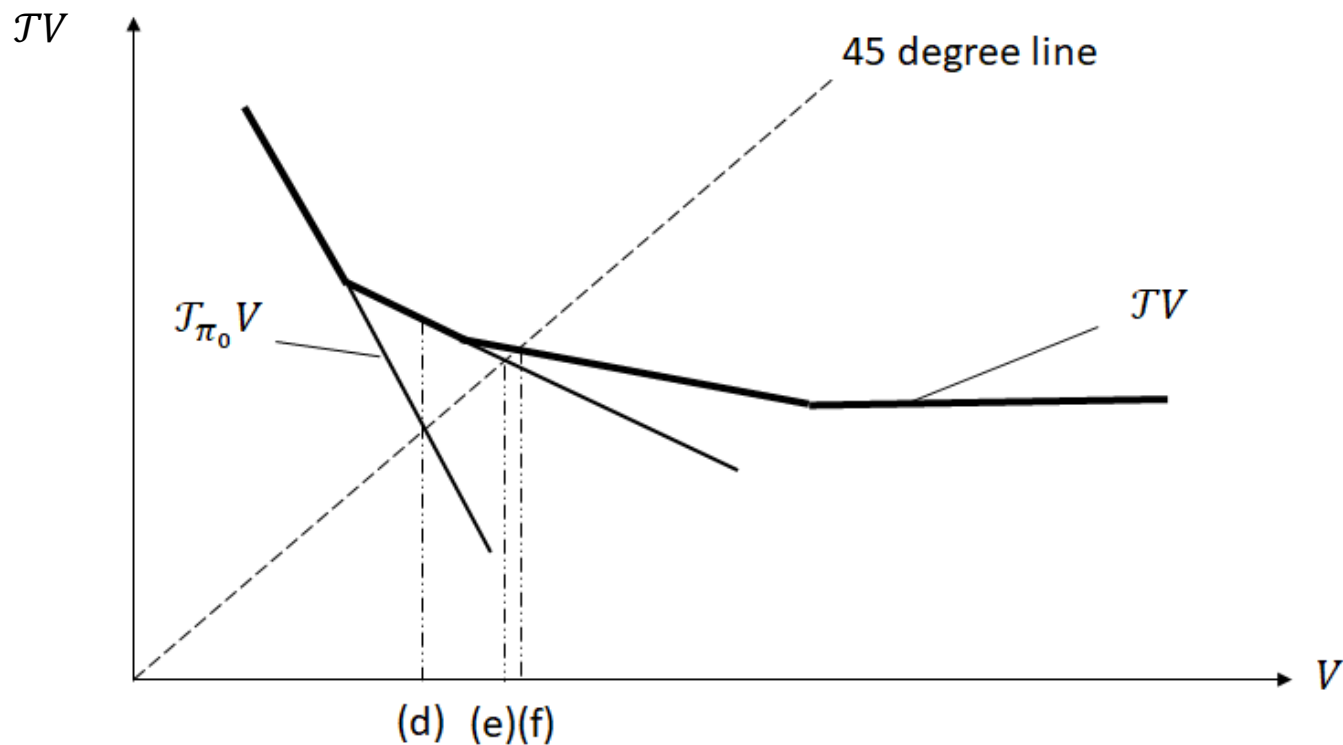
$V^\pi$  at three iterations (converged)

(d)

# Value iteration: geometric Interpretation

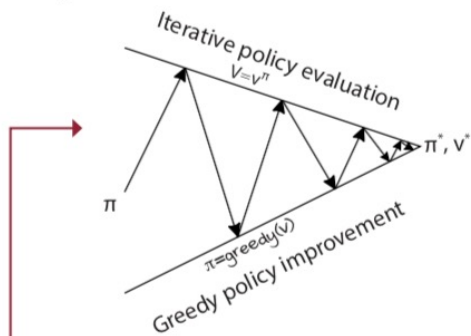


# Policy iteration: geometric Interpretation



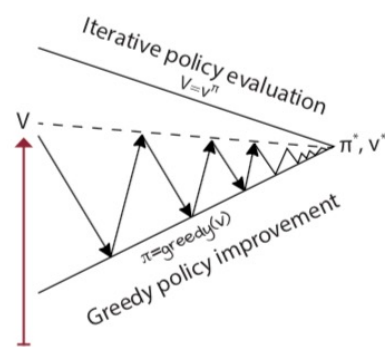
# More variations

## Policy iteration



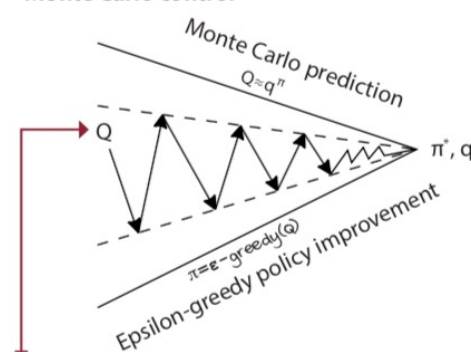
(1) Policy iteration consists of a full convergence of iterative policy evaluation alternating with greedy policy improvement.

## Value iteration



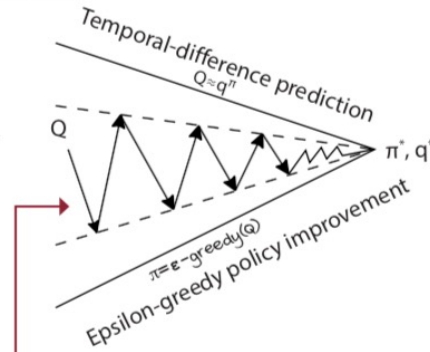
(2) value iteration starts with an arbitrary value function and has a truncated policy evaluation step.

## Monte Carlo control



(3) MC control estimates a Q-function, has a truncated MC prediction phase followed by an epsilon-greedy policy-improvement step.

## SARSA



(4) SARSA has pretty much the same as MC control except a truncated TD prediction for policy evaluation.

# Summary & Takeaways

- When specifying a sequential problem, care should be taken to select an appropriate type of policy and value function, **depending on the use case**.
- The ideas from **dynamic programming**, namely the **principle of optimality**, carry over to **infinite horizon** problems.
- The **value iteration** algorithm solves discounted infinite horizon MDP problems by leveraging results of **Bellman operators**, namely the **optimal Bellman equation**, **contractions**, and **fixed points**.
- **Generalized policy iteration** methods include policy iteration and value iteration.
- **Policy iteration** algorithm additionally leverages **monotonicity** and **Bellman equation**.
- The update mechanism for VI and PI differ and thus their convergence in practice depends on the **geometric structure** of the optimal value function.